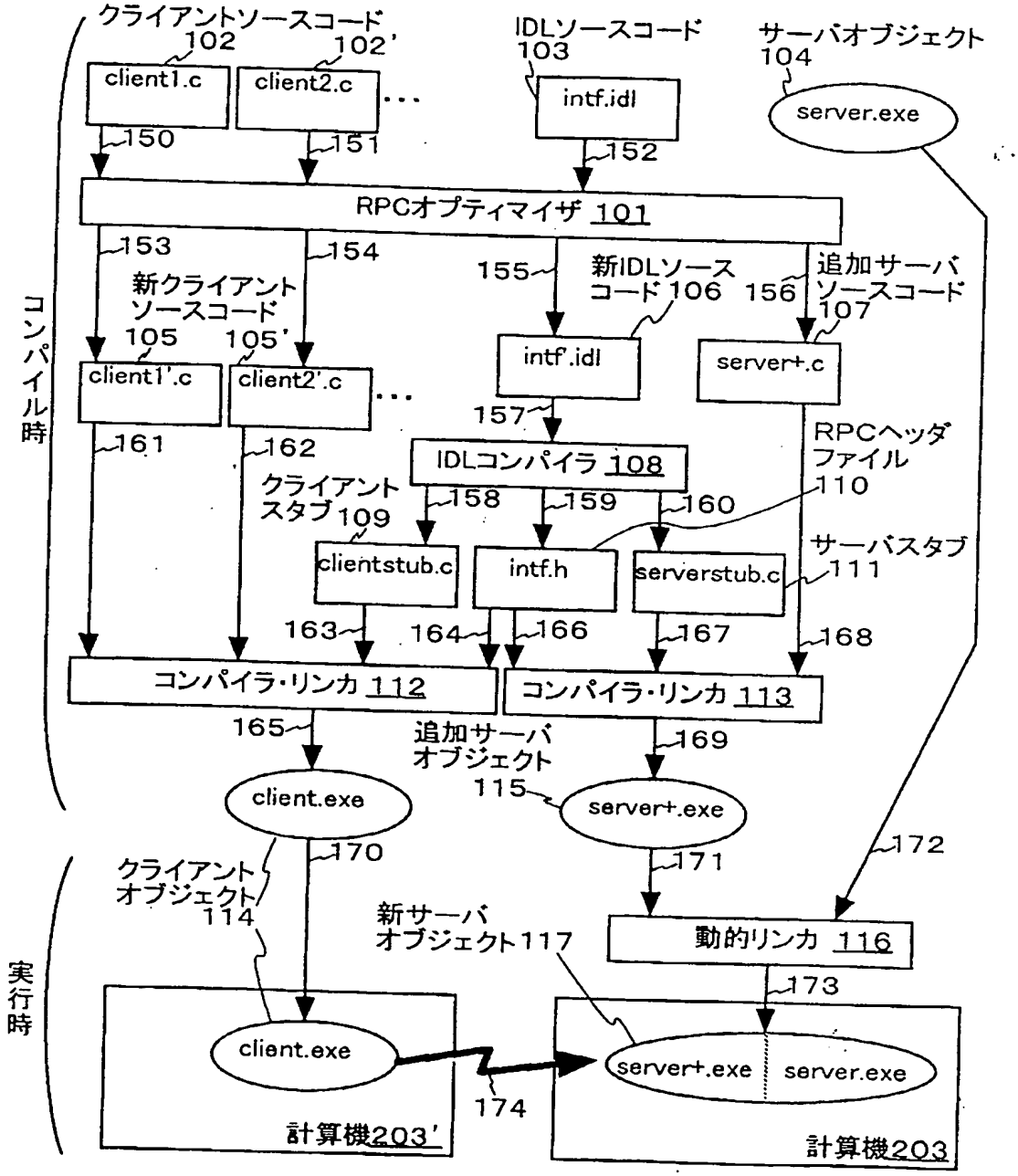
【書類名】　図面

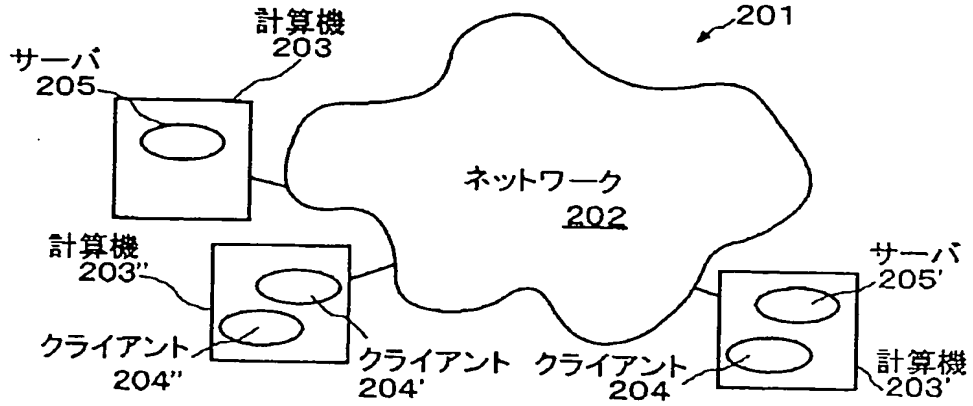
【図１】

図１


クライアントソースコード
102
102'
client1.c
client2.c
IDLソースコード
103
intf.idl
サーバオブジェクト
104
server.exe
150
151
152
RPCオプティマイザ 101
153
154
155
新IDLソースコード 106
156
追加サーバソースコード
107
新クライアントソースコード
105
105'
client1'.c
client2'.c
intf'.idl
server+.c
コンパイル時
157
RPCヘッダファイル
110
161
162
IDLコンパイラ 108
クライアントスタブ 109
158
159
160
clientstub.c
intf.h
serverstub.c
サーバスタブ
111
163
164
166
167
168
コンパイラ・リンカ 112
コンパイラ・リンカ 113
165
追加サーバオブジェクト
169
115
client.exe
server+.exe
170
171
172
クライアントオブジェクト
114
新サーバオブジェクト 117
動的リンカ 116
173
実行時
client.exe
server+.exe
server.exe
174
計算機203'
計算機203

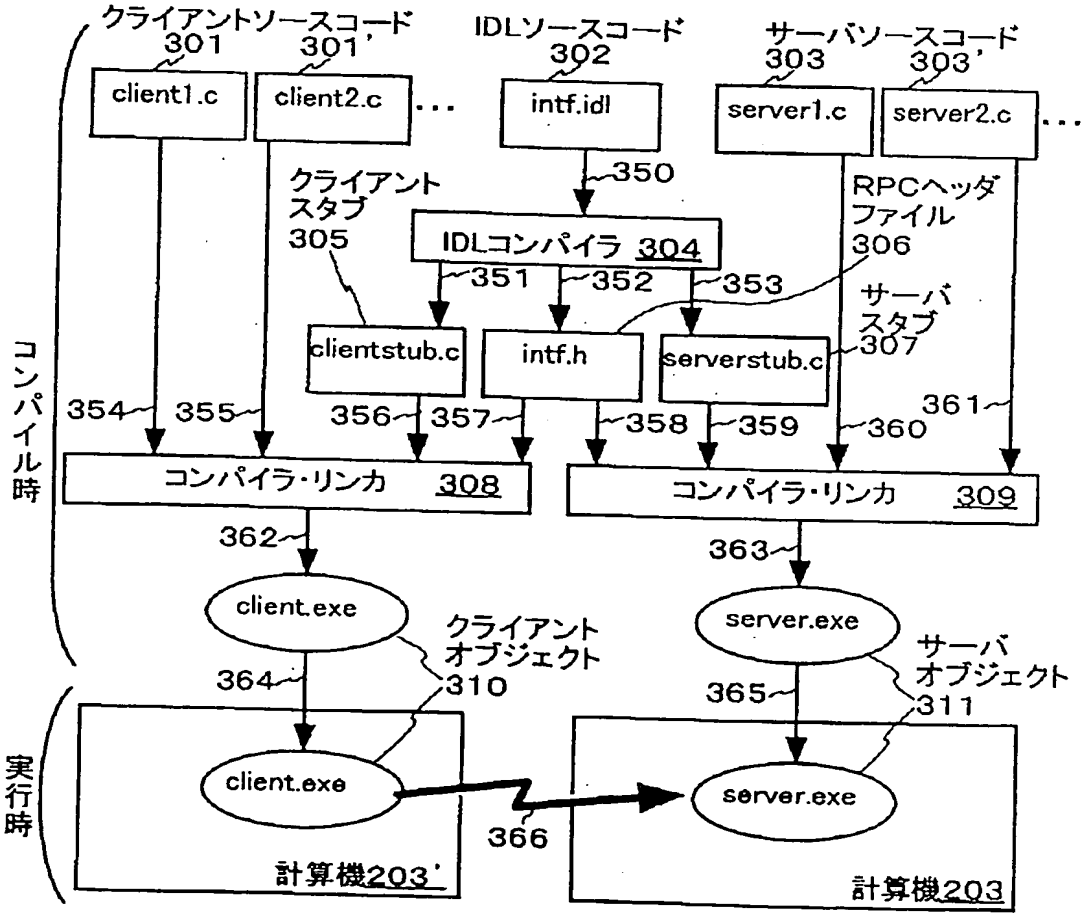
【図２】

図2


計算機
203
サーバ
205
201
ネットワーク
202
計算機
203''
サーバ
205'
クライアント
204''
クライアント
204'
クライアント
204
計算機
203'

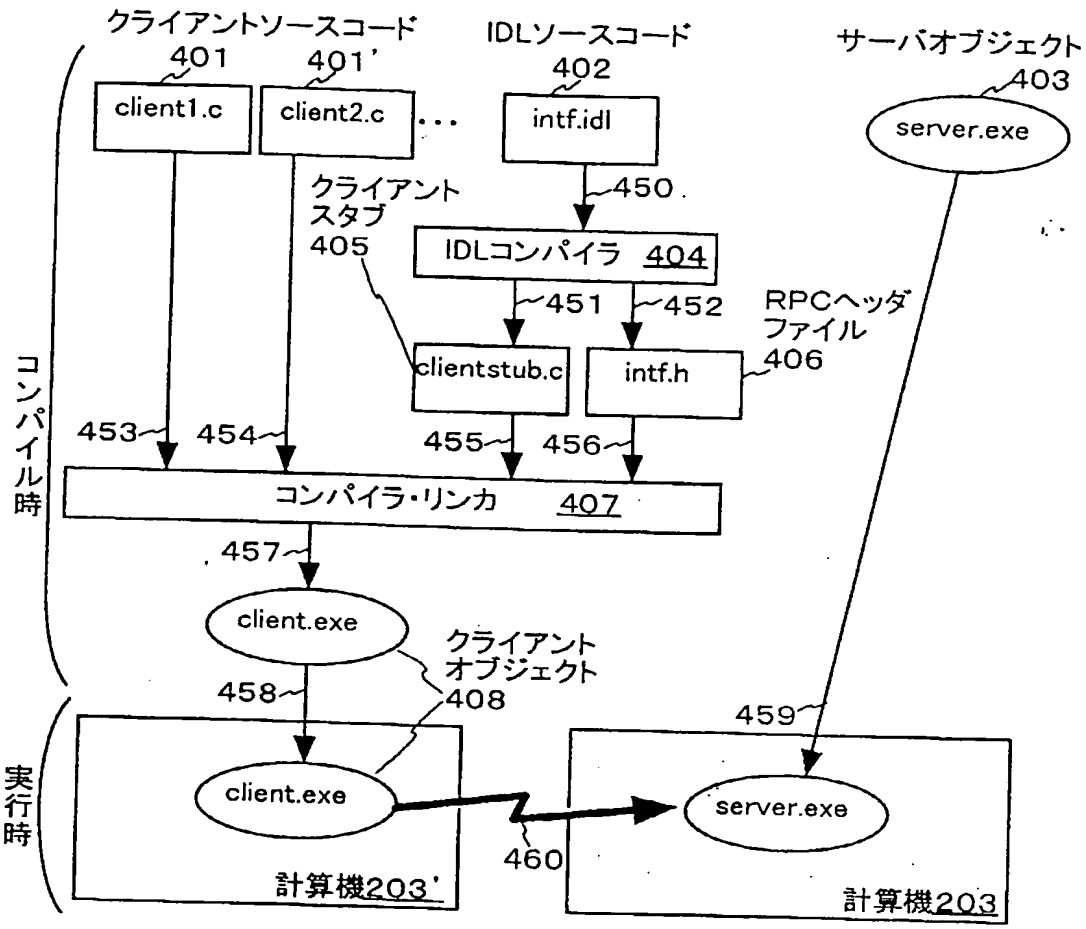
【図３】

図3


クライアントソースコード
301
301'
IDLソースコード
302
サーバソースコード
303
303'
client1.c
client2.c
intf.idl
server1.c
server2.c
350
クライアント
スタブ
305
IDLコンパイラ 304
RPCヘッダ
ファイル
306
351
352
353
サーバ
スタブ
307
clientstub.c
intf.h
serverstub.c
354
355
356
357
358
359
360
361
コンパイル時
コンパイラ・リンカ 308
コンパイラ・リンカ 309
362
363
client.exe
server.exe
クライアント
オブジェクト
310
サーバ
オブジェクト
311
364
365
実行時
client.exe
server.exe
366
計算機203'
計算機203

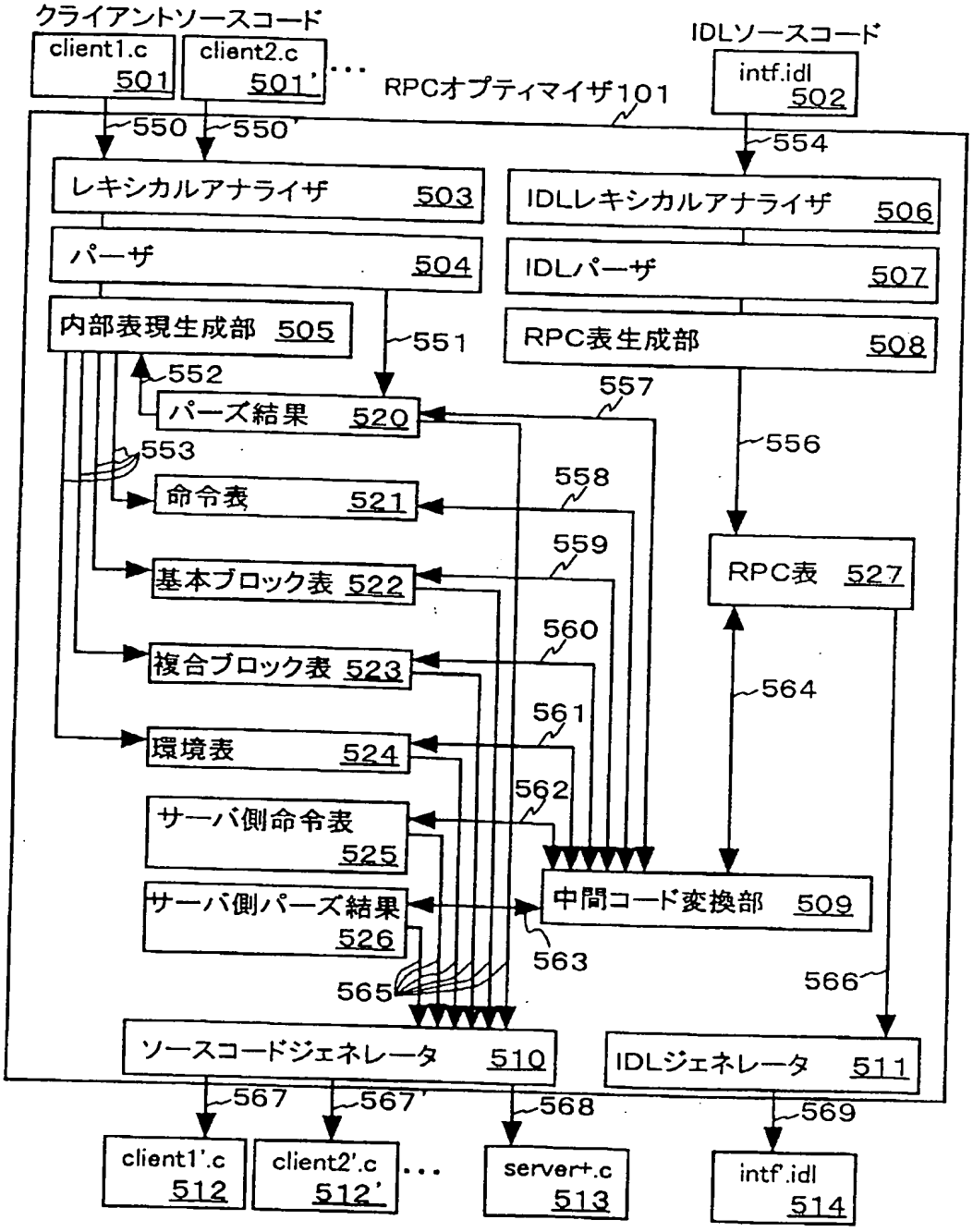
【図４】

図4


クライアントソースコード
401
401'
client1.c
client2.c
...
IDLソースコード
402
intf.idl
サーバオブジェクト
403
server.exe
クライアント
スタブ
405
450
IDLコンパイラ 404
451
452
RPCヘッダ
ファイル
406
clientstub.c
intf.h
コンパイル時
453
454
455
456
コンパイラ・リンカ 407
457
client.exe
クライアント
オブジェクト
408
458
459
実行時
client.exe
server.exe
460
計算機203'
計算機203

【図５】

# 図5


クライアントソースコード
client1.c 501
client2.c 501'
RPCオプティマイザ101
IDLソースコード
intf.idl 502
550
550'
554
レキシカルアナライザ 503
IDLレキシカルアナライザ 506
パーザ 504
IDLパーザ 507
内部表現生成部 505
551
RPC表生成部 508
552
557
パーズ結果 520
553
556
558
命令表 521
559
RPC表 527
基本ブロック表 522
560
複合ブロック表 523
564
561
環境表 524
562
サーバ側命令表 525
サーバ側パーズ結果 526
中間コード変換部 509
563
565
566
ソースコードジェネレータ 510
IDLジェネレータ 511
567
567'
568
569
client1'.c 512
client2'.c 512'
server+.c 513
intf'.idl 514

【図６】

# 図6

命令表 600

| 命令ID 602 | ターゲット 603 | 命令 604 | オペランドA 605 | オペランドB 606 |
|---|---|---|---|---|
| 命令要素 601 | ⋮ | | | |

基本ブロック表 610

| 基本ブロックID 612 | 開始命令ID 613 | 終端命令ID 614 | |
|---|---|---|---|
| 次基本ブロック 615 | 前基本ブロック 616 | 環境ID 617 | |
| DGEN変数表 618 | DKILL変数表 619 | DIN変数表 620 | DOUT変数表 621 |
| LIN変数表 622 | LOUT変数表 623 | LUSE変数表 624 | LDEF変数表 625 |
| 基本ブロック要素 611 | ⋮ | | |

複合ブロック表 630

| 複合ブロックID 632 | 開始基本ブロックID 633 | 終端基本ブロックID 634 | 環境ID 635 |
|---|---|---|---|
| 複合ブロック要素 631 | ⋮ | | |

環境表 640

| 環境ID 641 | 親環境ID 642 | 属性 643 |
|---|---|---|
| 環境内変数表 644 | | |

RPC表 650

| RPC名 652 | IN引数表 653 | OUT引数表 654 | 属性 655 |
|---|---|---|---|
| RPC表要素 651 | ⋮ | | |

| 型名 656 | 型情報 657 |
|---|---|
| 型宣言要素 658 | ⋮ |

変数表 660

| 変数名 662 | 型 663 | 属性 664 |
|---|---|---|
| 変数表要素 661 | ⋮ | |

【図７】

# 図7

intf.idl

```
701 interface MyServer {
702     int func1(in int i);
703     void func2(inout long key, in String value);
704 };
```

700

client1.c

```
751 #include "intf.h"

752 main()
753 {
754     MyServer server = lookupDirectory("MyServer");
755     int count = 0;
756     for (int i = 0; i < 100; i++)
757         count += server.func1(i);
758     }
759     printf("count=%d¥n", count);
760     server.func2(100, "hello world");
761     server.func1(j);
762 }
```

750

【図８】

# 図8

intf.h

```
801 #include "Object.h"

802 class MyServer: public Object {
803     int func1(int i);
804     void func2(long& key, char* value);
805 }
```

800

clientstub.c

```
851 #include "intf.h"

852 int MyServer::func1(int i)
853 {
854     Buffer buf = new Buffer();
855     int rval;
856     buf.packint(i);
857     call("func1", buf);
858     buf.unpackint(&rval);
859     delete buf;
860     return rval;
861 }

862 void MyServer::func2(long& key, char* value)
863 {
864     Buffer buf = new Buffer();
685     buf.packlong(key);
866     buf.packString(value);
867     call("func2", buf);
868     buf.unpacklong(&key);
869     delete buf;
870 }
```

850

【図９】

# 図9

serverstub.c　900

```
901 #include "intf.h"

902 void MyServer::loop()
903 {
904     while (1) {
905         Buffer buf;
906         Client client;
907         receive(&client, &buf);
908         if (buf.method.equals("func1")) {
909             int i, rval;
910             buf.unpackint(&i);
911             rval = func1(i);
912             buf.packint(rval);
913         } else if (buf.method.equals("func2") {
914             long key;
915             char* value;
916             buf.unpacklong(&key);
917             buf.unpackString(&value);
918             func2(key, value);
919             buf.packlong(key);
920         } else {
921             send(client, "error");
922             continue;
923         }
924         send(client, buf);
925         delete buf;
926         delete client;
927     }
928 }
```

【図１０】

# 図10

intf.idl

```
1001 interface MyServer {
1002     int func1(in int i);
1003     void func2(inout long key, in String value);
1004     void func3(inout int count);
1005     void func4(in int i);
1006 };
```

1000

client1.c

```
1011 #include "intf.h"

1012 main()
1013 {
1014     MyServer server = lookupDirectory("MyServer");
1015     int count = 0;
1016     server.func3(count);
1017     printf("count=%d¥n", count);
1018     server.func4(j);
1019 }
```

1010

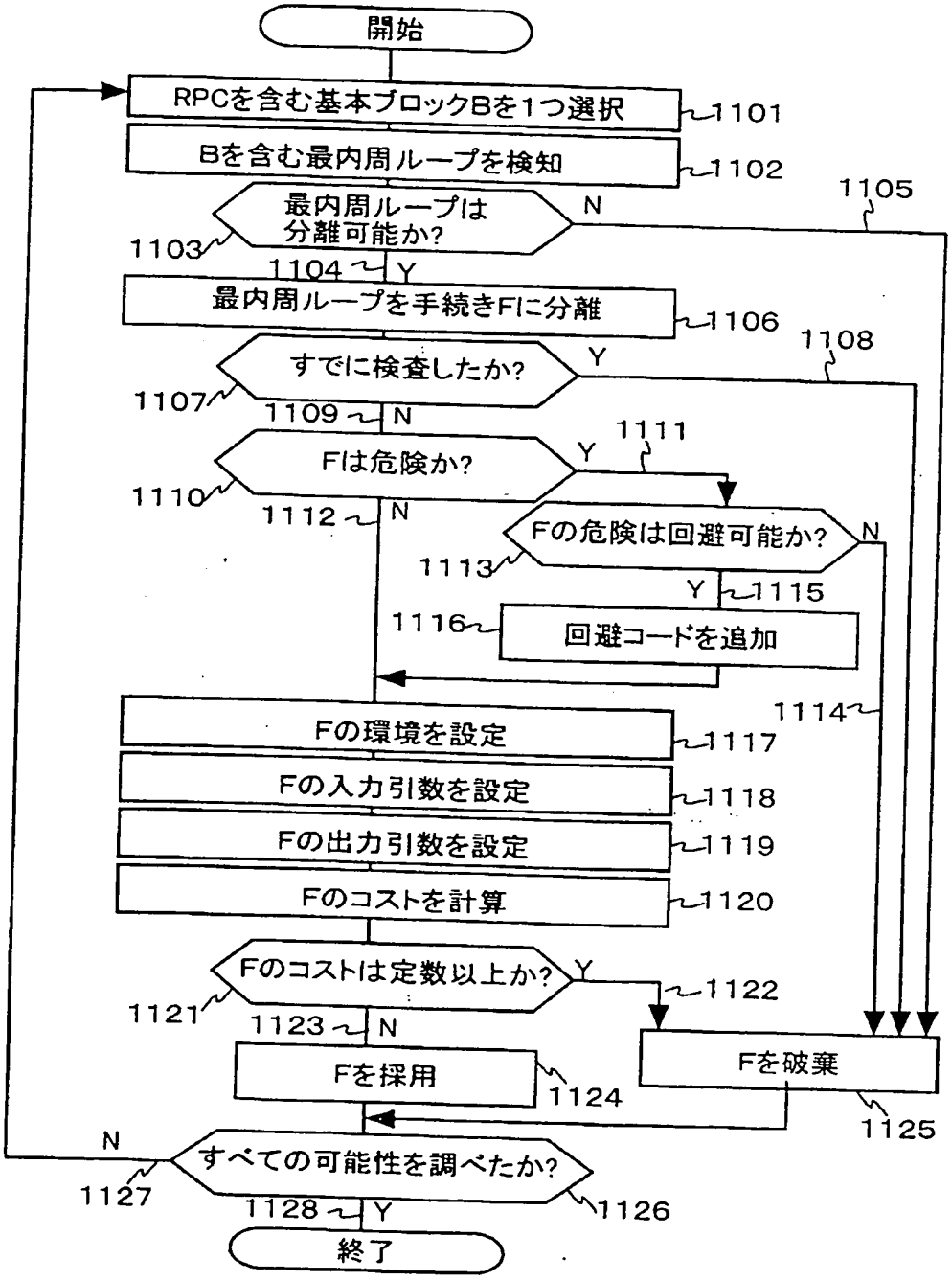
server+.c

```
1031 #include "intf.h"

1032 void MyServer::func3(int& count)
1033 {
1034     for (int i = 0; i < 100; i++)
1035         count += server.func1(i);
1036 }

1037 void MyServer::func4(lint count)
1038 {
1039     server.func2(100, "hello world");
1040     server.func1(count);
1041 }
```

1030

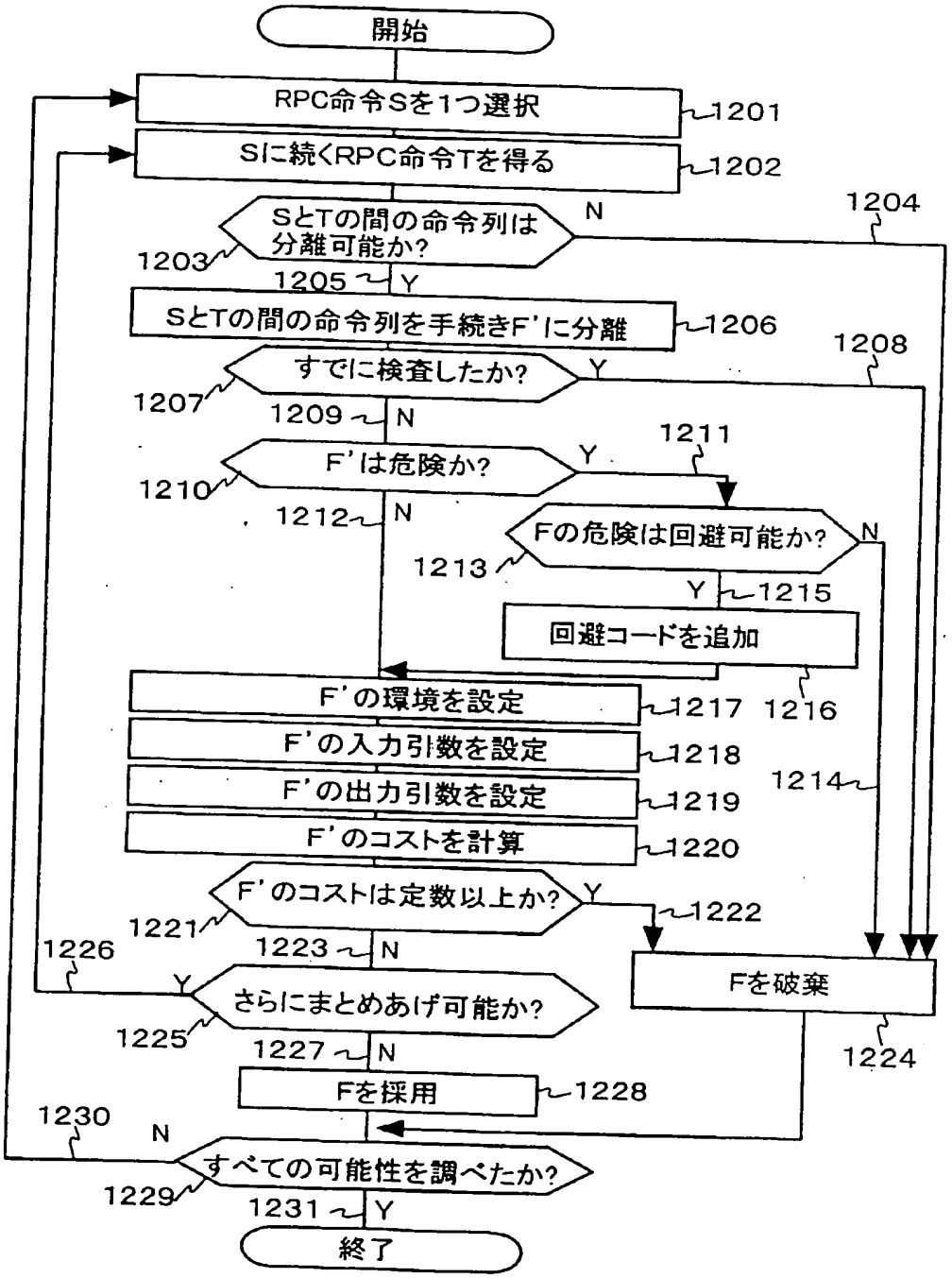
【図１１】

図11


開始
RPCを含む基本ブロックBを1つ選択
1101
Bを含む最内周ループを検知
1102
最内周ループは分離可能か?
1103
N
1105
Y
1104
最内周ループを手続きFに分離
1106
すでに検査したか?
1107
Y
1108
N
1109
Fは危険か?
1110
Y
1111
N
1112
Fの危険は回避可能か?
1113
N
Y
1115
回避コードを追加
1116
Fの環境を設定
1114
1117
Fの入力引数を設定
1118
Fの出力引数を設定
1119
Fのコストを計算
1120
Fのコストは定数以上か?
1121
Y
1122
N
1123
Fを採用
1124
Fを破棄
1125
すべての可能性を調べたか?
1126
N
1127
Y
1128
終了

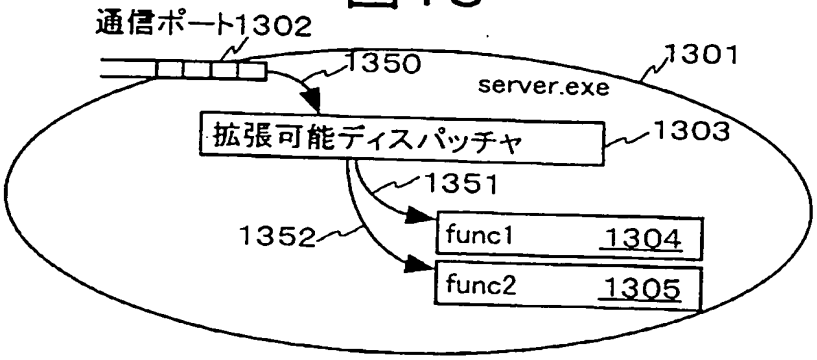
【図１２】

図12


開始
RPC命令Sを1つ選択
1201
Sに続くRPC命令Tを得る
1202
SとTの間の命令列は分離可能か?
1203
N
1204
Y
1205
SとTの間の命令列を手続きF'に分離
1206
すでに検査したか?
1207
Y
1208
N
1209
F'は危険か?
1210
Y
1211
N
1212
Fの危険は回避可能か?
1213
N
Y
1215
回避コードを追加
1216
F'の環境を設定
1217
F'の入力引数を設定
1218
F'の出力引数を設定
1219
F'のコストを計算
1220
1214
F'のコストは定数以上か?
1221
Y
1222
N
1223
Fを破棄
1224
さらにまとめあげ可能か?
1225
Y
1226
N
1227
Fを採用
1228
すべての可能性を調べたか?
1229
N
1230
Y
1231
終了

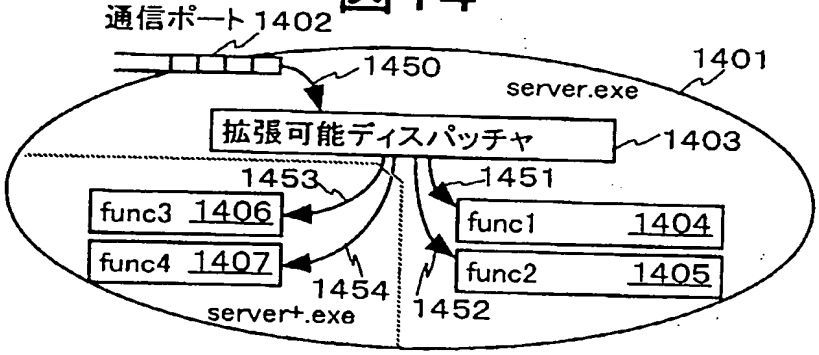
【図１３】

図13



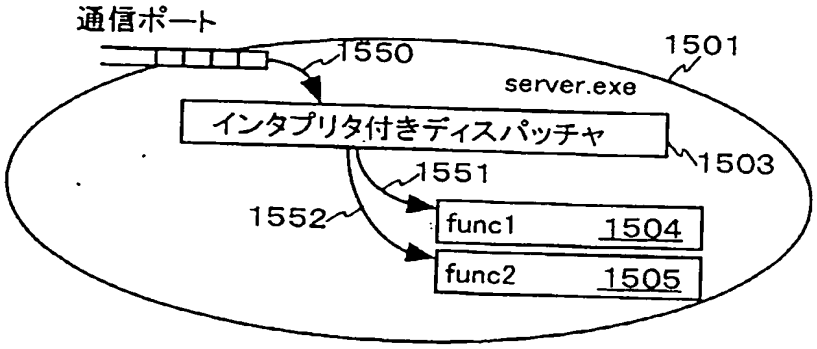

【図１４】

図14



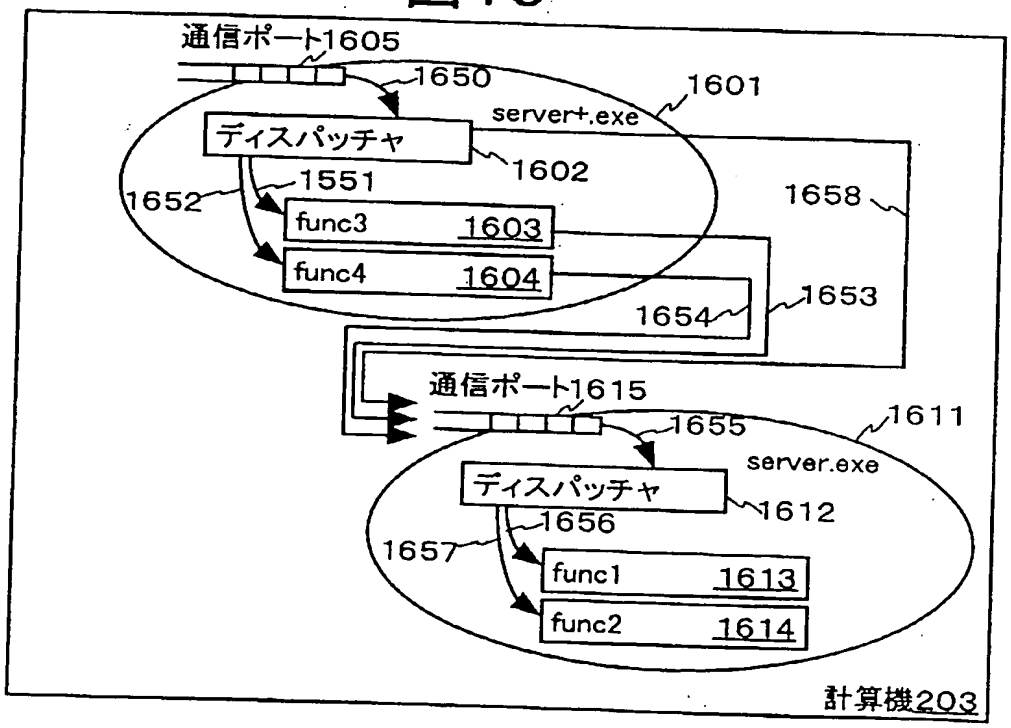
【図１５】

図15



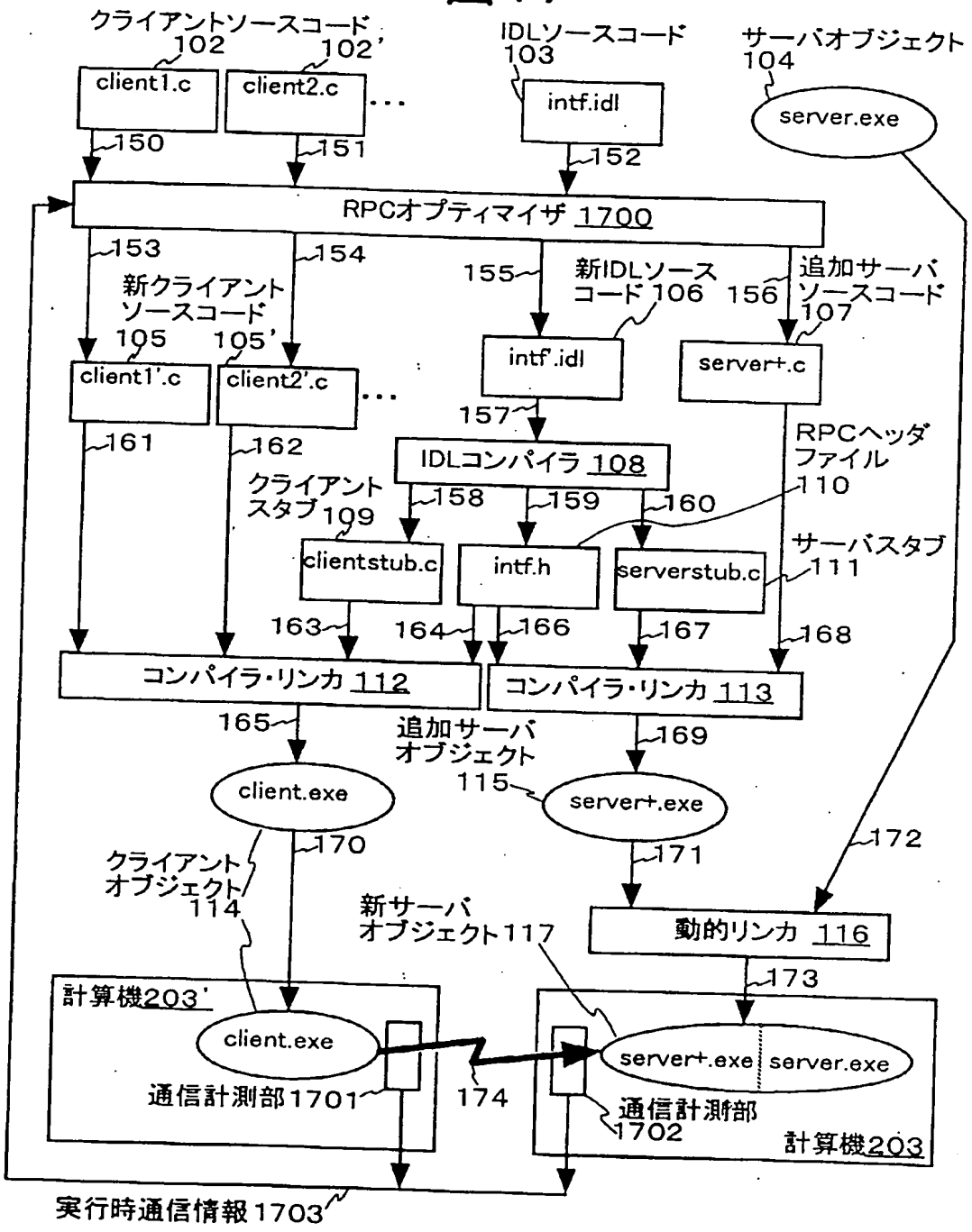

【図１６】

図16



【図１７】

図17


クライアントソースコード
102
client1.c
102'
client2.c
IDLソースコード
103
intf.idl
サーバオブジェクト
104
server.exe
150
151
152
RPCオプティマイザ 1700
153
154
155
新IDLソースコード 106
156
追加サーバソースコード
107
新クライアントソースコード
105
105'
client1'.c
client2'.c
intf'.idl
server+.c
157
RPCヘッダファイル
110
161
162
IDLコンパイラ 108
クライアントスタブ 109
158
159
160
clientstub.c
intf.h
serverstub.c
サーバスタブ
111
163
164
166
167
168
コンパイラ・リンカ 112
コンパイラ・リンカ 113
165
追加サーバオブジェクト
169
client.exe
115
server+.exe
170
171
172
クライアントオブジェクト
114
新サーバオブジェクト 117
動的リンカ 116
173
計算機203'
client.exe
server+.exe
server.exe
通信計測部 1701
174
通信計測部
1702
計算機203
実行時通信情報 1703

【図１８】

# 図18

extended_intf.idl

```
1801  interface MyServer {
1802      int func1(in int i) const;
1803      void func2(inout long key, in String value);
1804      int func3(void);

1805      commutative { func2, func3 };

1806      parallel { func1, func2, func3 };
1807  };
```

1800

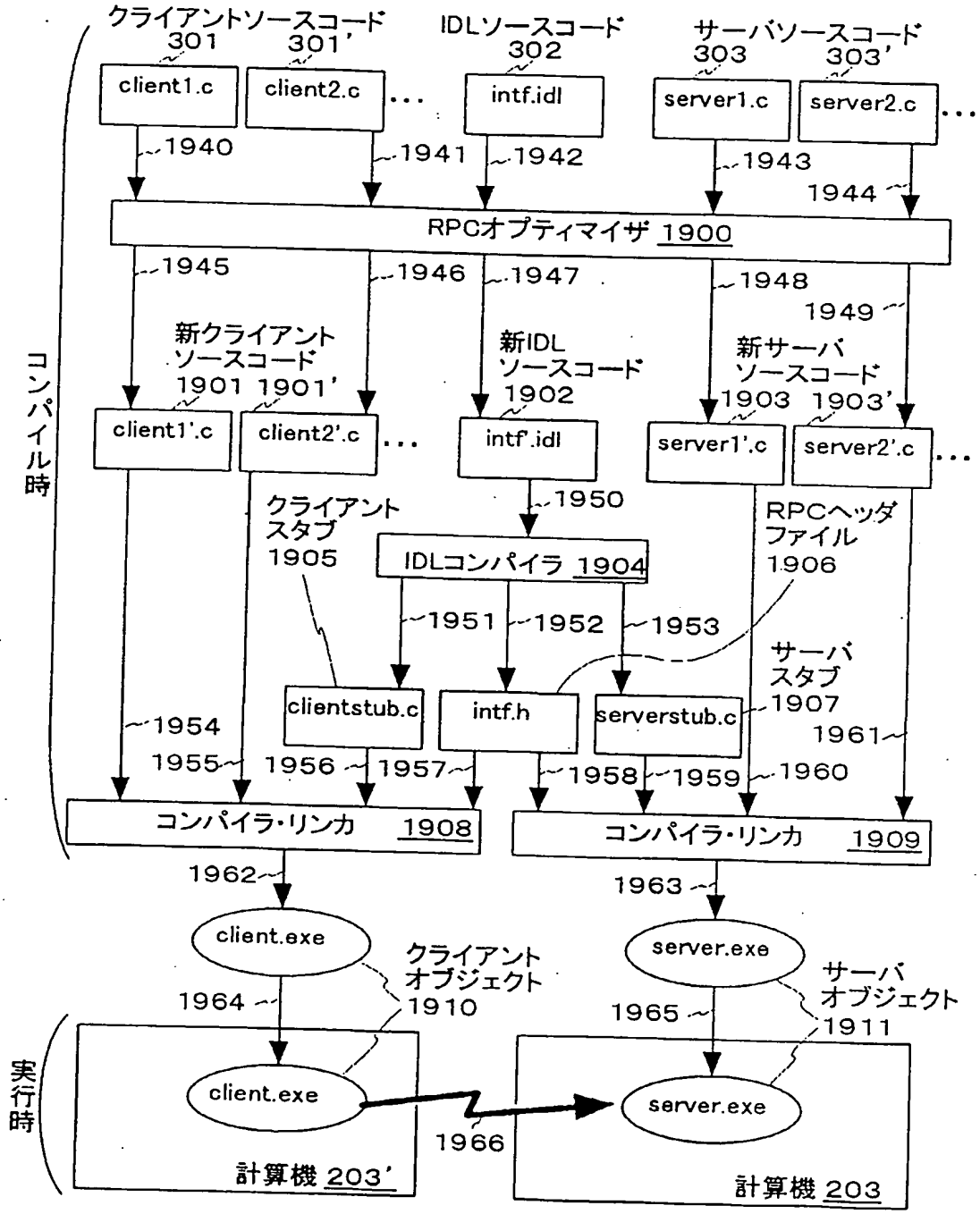
server+.c

```
1821  #include "intf.h"
1822  #include "thread.h"

1823  void MyServer::func3(int& count)
1824  {
1825      List<Thread> allThreads;
1826      Thread t;
1827      void *rval;
1828      for (int i = 0; i < 100; i++) }
1829          create_thread(&t, server.func1, 1, i),;
1830          allThreads.add(t);
1831      }
1832      for ( ; (t = allThreads.next()) != NULL_THREAD; ) {
1833          join_thread(t, &rval);
1834          count += *(int *)rval;
1835      }
1836  }

1837  void MyServer::func4(lint count)
1838  {
1839      List<Thread> allThreads;
1840      Thread t;
1841      create_thread(&t, server.func2, 2, 100, "hello world");
1842      allThreads.add(t);
1843      create_thread(&t, server.func1, 1, count);
1844      allThreads.add(t);
1845      for ( ; (t = allThreads.next()) != NULL_THREAD; )
1846          join_thread(t, NULL);
1847  }
```

1820

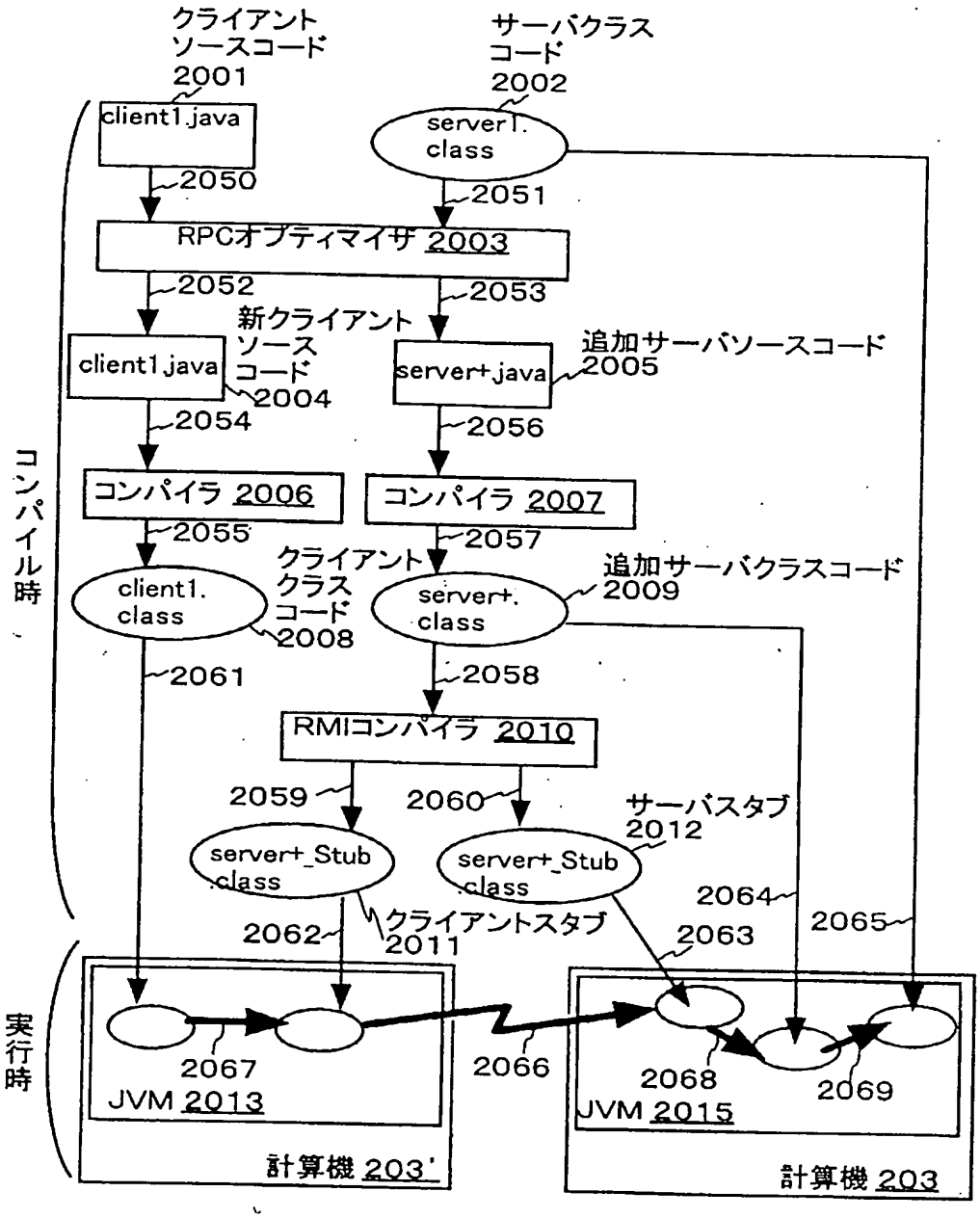
【図１９】

図19


クライアントソースコード
301
301'
IDLソースコード
302
サーバソースコード
303
303'
client1.c
client2.c
intf.idl
server1.c
server2.c
1940
1941
1942
1943
1944
RPCオプティマイザ 1900
1945
1946
1947
1948
1949
新クライアント
ソースコード
1901 1901'
新IDL
ソースコード
1902
新サーバ
ソースコード
1903 1903'
コンパイル時
client1'.c
client2'.c
intf'.idl
server1'.c
server2'.c
1950
クライアント
スタブ
1905
IDLコンパイラ 1904
RPCヘッダ
ファイル
1906
1951
1952
1953
サーバ
スタブ
1907
clientstub.c
intf.h
serverstub.c
1954
1955
1956
1957
1958
1959
1960
1961
コンパイラ・リンカ 1908
コンパイラ・リンカ 1909
1962
1963
client.exe
server.exe
クライアント
オブジェクト
1910
サーバ
オブジェクト
1911
1964
1965
実行時
client.exe
server.exe
1966
計算機 203'
計算機 203

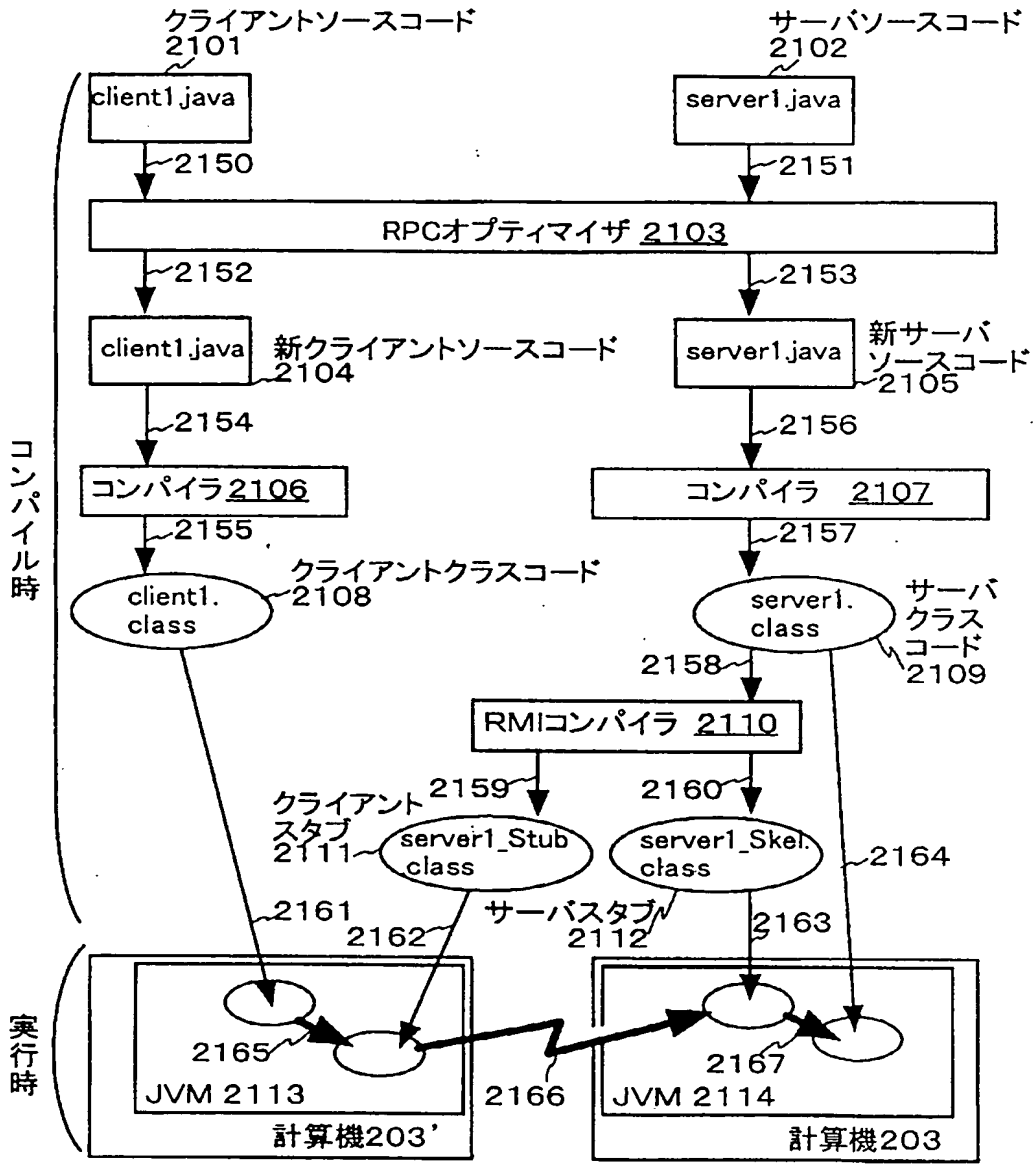
【図２０】

図20


クライアント
ソースコード
2001
サーバクラス
コード
2002
client1.java
server1.
class
2050
2051
RPCオプティマイザ 2003
2052
2053
新クライアント
ソース
コード
2004
client1.java
server+.java
追加サーバソースコード
2005
2054
2056
コンパイル時
コンパイラ 2006
コンパイラ 2007
2055
2057
クライアント
クラス
コード
2008
client1.
class
server+.
class
追加サーバクラスコード
2009
2061
2058
RMIコンパイラ 2010
2059
2060
サーバスタブ
2012
server+_Stub
.class
server+_Stub
.class
2064
2062
クライアントスタブ
2011
2063
2065
実行時
2067
JVM 2013
2066
2068
JVM 2015
2069
計算機 203'
計算機 203

【図２１】

図21


クライアントソースコード
2101
サーバソースコード
2102
client1.java
server1.java
2150
2151
RPCオプティマイザ 2103
2152
2153
client1.java
新クライアントソースコード
2104
server1.java
新サーバソースコード
2105
2154
2156
コンパイラ2106
コンパイラ 2107
2155
2157
クライアントクラスコード
2108
client1. class
server1. class
サーバクラスコード
2109
2158
RMIコンパイラ 2110
2159
2160
クライアントスタブ
2111
server1_Stub class
server1_Skel. class
2164
2161
2162
サーバスタブ
2112
2163
2165
JVM 2113
2166
2167
JVM 2114
計算機203'
計算機203
コンパイル時
実行時